RAI: An LLM-Based Multi-Agent Tool for Requirements Engineering Artifacts Generation

Guilherme Borges Cabral
LEDS Research Group, Federal Institute of Espírito Santo
Serra-ES, Brazil
guilhermeborgescabral@gmail.com

Nicolas Duarte Botelho
LEDS Research Group, Federal Institute of Espírito Santo
Serra-ES, Brazil
nicolasbotelho2005@gmail.com

Victorio Albani Carvalho
LEDS Research Group, Federal Institute of Espírito Santo
Serra-ES, Brazil
victorio@ifes.edu.br

Moisés Savedra Omena
LEDS Research Group, Federal Institute of Espírito Santo
Serra-ES, Brazil
omenam@ifes.edu.br

Rodrigo Calhau
LEDS Research Group, Federal Institute of Espírito Santo
Serra-ES, Brazil
calhau@ifes.edu.br

Paulo Sérgio dos Santos Júnior
LEDS Research Group, Federal Institute of Espírito Santo
Serra-ES, Brazil
paulo.junior@ifes.edu.br

Monalessa P. Barcellos
Ontology & Conceptual Modeling Research Group (NEMO), Federal University of Espírito Santo
Vitória-ES, Brazil
monalessa@inf.ufes.br

Abstract

Requirements Engineering (RE) plays a central role in software development. Eliciting and specifying requirements remains a challenging and time-consuming activity, as it involves interpreting complex, often tacit stakeholder needs and translating them into artifacts that are understandable to clients and actionable by the development team. Recent advances in Large Language Models (LLMs) and LLM-based agents offer new opportunities to support RE activities. This paper presents RAI (*Requirements AssIstant*), an LLM-based multi-agent tool that generates RE artifacts from audio or video recordings of elicitation sessions. RAI progressively produces a domain storytelling; a requirements document comprising functional requirements, business rules, and non-functional requirements; a use case diagram with corresponding use case descriptions, and a class diagram accompanied by a data dictionary. The tool follows a human-in-the-loop approach, allowing the requirements engineer to revise and edit generated artifacts and use the revised artifacts as input for generating subsequent artifacts. RAI also supports artifact versioning, maintains traceability among artifacts, and identifies potential inconsistencies. Preliminary results from using RAI in four organizations indicate that it reduced time and effort in requirements documentation and supported the production of artifacts with good accuracy, coverage, and relevance.
Demo video: https://figshare.com/s/6ec5e7ad9836d3398677

Keywords

Requirement, LLM, Agent, Tool, Use Case, Class Diagram

1 Introduction

The increasing complexity and dynamism of modern digital society have impacted software development. As organizations face growing expectations for agility, customization, and innovation, software engineers are required to cope with accelerated delivery cycles, frequent requirement changes, and evolving business models. This scenario demands the ability to continuously adapt processes and tools to meet shifting organizational and user needs [4].

These changes not only affect software implementation and delivery practices but also drive the evolution of traditionally well-established areas, such as Requirements Engineering (RE) [7, 10]. RE provides crucial information for software construction, project management, and strategic decision-making. A failure in this process can result in misunderstandings that propagate throughout the development cycle. In this context, requirements elicitation and specification are of paramount importance. They require understanding complex and often implicit stakeholder needs, translating them into artifacts that are accessible to clients and useful for software developers [10]. Traditional RE approaches are often time-consuming, resource-intensive, and highly dependent on formal documentation and structured communication methods [14, 19].

Recent advancements in Artificial Intelligence (AI) and the popularization of Large Language Models (LLMs) have led to some initiatives to aid in RE (e.g., [12, 15]). Such advances have also catalyzed the development of a new class of intelligent systems known as LLM-based agents. They integrate natural language understanding, reasoning, and generative capabilities of LLMs with mechanisms for perception, planning, memory, and autonomous decision-making. Unlike conventional LLMs, which typically operate reactively, generating responses based solely on static prompts, LLM-based agents exhibit proactive and adaptive behavior [21, 27]. They are capable of interacting dynamically with both digital and physical environments, orchestrating multi-step tasks, utilizing external tools and APIs, and maintaining internal states through short- and long-term memory architectures [20, 26]. This marks a

paradigm shift: from LLMs as passive assistants to cognitive agents capable of simulating complex human behaviors and workflows.

In this emerging context, LLM-based agents have demonstrated great potential to support several Software Engineering (SE) activities. Their architectural advantage lies in the ability to synthesize heterogeneous information, perform reflective reasoning, and adapt their strategies through iterative feedback loops[16]. Unlike prompt-based approaches, which usually rely on isolated user instructions and single-step responses, LLM-based agents can decompose complex tasks, coordinate intermediate reasoning steps, and maintain goal-oriented workflows across multiple interactions.

This paper presents *RAI* (*Requirements AssIstant*), an LLM-based multi-agent tool that generates RE artifacts from audio or video inputs. By leveraging automatic speech recognition and natural language processing, RAI captures and interprets conversations recorded during requirements elicitation sessions, extracts relevant information, and structures it into a *domain storytelling*. Based on this narrative, *RAI* generates a *requirements document* comprising functional requirements (FR), business rules (BR), and non-functional requirements (NFR), plus questions aimed at assisting the requirements engineer in improving domain understanding and clarifying or refining requirements. Finally, from the requirements document, RAI derives a *use case* diagram with use case descriptions, and a *class diagram* accompanied by a data dictionary.

In line with recent human-in-the-loop approaches for LLM-based modeling, in which automatically generated models are refined through user feedback and interaction [25], RAI keeps the requirements engineer actively involved throughout artifact generation and refinement. In this sense, the tool does not impose a fully automated pipeline. Instead, it allows the requirements engineer to edit a generated artifact and request the generation of subsequent artifacts based on the edited version. For example, the requirements engineer may refine the domain storytelling generated by RAI before asking the tool to generate the requirements document, or they can modify the requirements document produced by RAI before asking the tool to generate use case and class diagrams. This interactive process enables the requirements engineer to incorporate domain knowledge, correct inaccuracies, and progressively improve the consistency and quality of the generated RE artifacts.

The main contribution of this paper is RAI. This work also contributes to the investigation of how LLM-based multi-agent systems can support RE by transforming unstructured elicitation data into a set of structured and interrelated RE artifacts. By combining artifacts generation and human-in-the-loop refinement, RAI aims to reduce the effort required to produce initial RE documentation, preserving the requirements engineer's role in validating, correcting, and improving the generated outputs.

2 Related Work

With the advancement of LLMs, studies have explored their applications in SE in general (e.g., [3, 6, 9, 11, 13, 24]) and in RE in particular, supporting tasks such as detecting incompleteness and inconsistencies in requirements [5] and teaching RE [15]. Most works have focused mainly on prompt-based solutions (i.e., the user interacts by prompting commands) or single agents. Moreover, the proposed solutions often receive structural data as input (such as legal or regulatory artifacts [1], technical documents or manuals [18], models or ontologies [17]) to generate artifacts, and address either requirements identification or modeling. For example, [2, 6] use LLMs to generate domain models from user stories. [23], in turn, uses LLMs to aid in requirements identification. RAI differs from these works by using a pipeline of LLM-based agents, addressing both requirements identification and modeling, and using unstructured data (audio or video files) as the input to generate the artifacts.

The work closest to ours is ReqGenie [12], which adopts a pipeline composed of a sequence of structured prompts, built on GPT, where the requirements engineer is guided step-by-step to provide information about the problem context, stakeholders, functional requirements, and non-functional requirements. This sequential interaction produces a requirements document that combines the user's inputs with AI-suggested content. Different from ReqGenie [12], RAI produces four artifacts, addressing different and complementary perspectives of the system. Moreover, in RAI, the initial information source are audio or video recordings. From these inputs, RAI automatically extracts, categorizes, structures and prioritizes requirements, and generates models addressing behavioral (use case) and structural (class diagram) perspectives. RAI also identifies points of attention, highlighting aspects that may require further clarification and improvement. By following a human-in-the-loop approach, RAI supports artifact refinement and keeps the requirements engineer responsible for the produced artifacts. Furthermore, different from the cited works, RAI manages the artifacts versions, traceability and consistency among them.

3 RAI's Requirements and Architecture

RAI's main purpose is to help the requirements engineer produce RE artifacts by transforming unstructured elicitation data into structured RE artifacts. To develop RAI, first, we established the tool's scope by defining its requirements. Then, we defined the tool's architecture, implemented it, and verified and validated it through testing. We followed an iterative process. In the first iteration, we developed the first version of the tool, which generated two artifacts. In the second iteration, we developed the second version, addressing four artifacts. Finally, in the third iteration, we addressed artifact version management and traceability, resulting in the current version of RAI.

Requirements. To guide RAI development, we defined six high-level requirements, which were thus decomposed into more specific ones. These requirements informed both the architectural decisions made and the features implemented in the tool. Next, we present the high-level requirements. The detailed requirements derived from them can be found in [8].

(R1) To materialize knowledge captured in requirements elicitation meetings, from elicitation unstructured data, RAI must support the *generation of RE artifacts* addressing different perspectives of the system to be developed. In this context, RAI must be able to process audio or video recordings, extract relevant information from them, and organize it into structured RE artifacts, aiming to accelerate initial requirements documentation.

(R2) RAI must help the requirements engineer *understand the domain* to be addressed by the system by structuring information

gathered in elicitation sessions, identifying relevant concepts, ambiguities, open issues, and potential inconsistencies that may emerge from elicited data. This aims to help the requirements engineer reflect on the captured information, prepare follow-up questions, and stimulate insights for subsequent requirements elicitation and refinement activities.

(R3) RAI must support *human intervention* in artifact creation and refinement. The requirements engineer must be allowed to modify automatically generated artifacts and use the revised version as input to generate subsequent ones. The requirements engineer remains responsible for the generated artifacts, and, thus, must be able to visualize, evaluate, and evolve them.

(R4) In RE, artifacts are progressively defined and refined so that each artifact deepens, reorganizes, or provides a different perspective of aspects captured in previous ones. Thus, RAI must maintain *traceability* between artifacts (e.g., which artifact was the source to generate another) and preserve *consistency* between them.

(R5) RAI must *manage different versions* of the artifacts and indicate potential outdated dependencies when source artifacts are modified. When a new version of an artifact is created, artifacts previously derived from the previous version of that artifact may no longer reflect the current state of the specification. Therefore, RAI must identify and communicate such situations.

(R6) Systems may address large or complex domains. Thus, RAI must support the *decomposition* of the system into subsystems/modules and the generation of RE artifacts to each of them.

General Architecture. RAI is a web-based tool in which specialized LLM-based agents cooperate to process elicitation data and generate RE artifacts. Figure 1 shows an overview of its architecture.

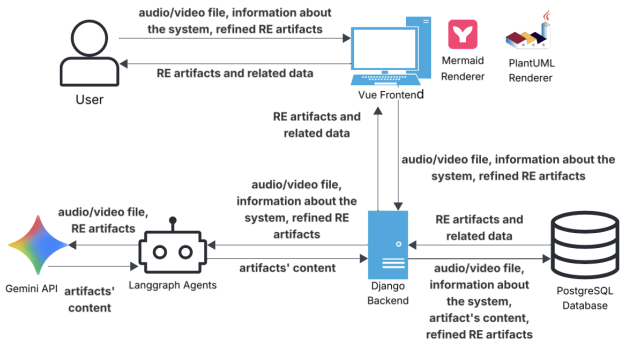

Figure 1: RAI's architecture.

The user interacts with RAI through a web-based front-end application built with Vue and Tailwind, following an MVC (Model-View-Controller) architecture. The RE artifacts managed by RAI are represented in plain-text formats, namely Markdown for textual artifacts, Mermaid for class diagrams, and PlantUML for use case diagrams. This choice allows the artifacts to be processable by AI agents in their original format, and to be rendered by the front-end to enhance the readability for requirements engineers.

The back-end was developed with Django, following a layered architecture, inspired by the MVC pattern, comprising controllers that handle incoming HTTP requests, models for data representation/persistence, and services for business logic. It exposes a RESTful API, documented using OpenAPI through a Django-OpenAPI/Swagger integration library. The architecture uses OAuth2 for application authentication and CORS to restrict access to the back-end API. Data persistence is handled by a PostgreSQL database, and Django's standard ORM is used to map model-layer entities to relational database structures.

The intelligence layer of RAI is implemented through LLM-based agents built with LangGraph and LangChain. These frameworks are used to define specialized agents and organize their execution into graphs according to their roles in the artifacts generation workflows. This layer communicates with the Django back-end through a graph invocation function that acts as an abstraction layer between the agent graphs and the application logic. Currently, RAI employs the gemini-3-flash-preview model, due to its support for multimodal inputs, large context windows, reasoning capabilities, and a favorable trade-off between latency and output quality. This decision was based on the model's technical capabilities and the resources available to the team, rather than on comparative benchmarks. The LLM provider can be replaced with limited implementation effort.

Agents Architecture. To create the agents, we used LangGraph, which enables declarative, graph-based workflows for language models. To each artifact to be produced, we created a graph defining the workflow to generate the respective artifact (namely, Domain Storytelling graph, Requirements graph, Use Case graph, and Class Diagram graph) and associated a set of agents with it, making each agent a node in the graph. An additional graph was created (Revision graph) to address artifacts' consistency. Together, these graphs contain 18 agents. This structure orchestrates the agents in a pipeline in which each graph uses as input the output produced by the previous one. The same occurs inside a graph, i.e., each agent produces an output that is used by the next agent as the input.

In a nutshell, from a file containing the transcription of an elicitation session, the *Domain Storytelling* graph's agents derive a structured domain narrative (domain storytelling) that consolidates information in natural language, avoiding technical jargon, and provides the source required for requirements identification. Thus, the *Requirements* graph's agents receive the domain storytelling, interpret it, and generate a structured requirements document encompassing FRs, BRs, NFRs, and open questions. The requirements document is used as input by the *Use Case* graph, whose agents create the use case diagram and corresponding descriptions. Based on that artifact, the *Class Diagram* graph's agents produce the class diagram and data dictionary. Finally, the *Review* graph's agents review the produced outputs to ensure consistency.

To exemplify how agents are organized in the graphs, Figure 2 illustrates the four agents involved in the Requirements graph. In the *Requirements* graph, the *Analyze Domain Narrative* agent (A1) receives as input the domain storytelling produced in the previous graph. Thus, it identifies FRs, BRs, and NFRs, as well as issues, ambiguities, and open questions, producing a requirements draft. These elements are classified and documented by the *Extract Requirements* agent (A2), informing, for each requirement, its id, description, suggested priority, and related requirements. The *Review Priorities* agent (A3) reviews the requirements priorities, aiming at

coherence and consistency. Finally, the *Refine Requirements* agent (A4) produces the final version of the requirements document in Markdown format.

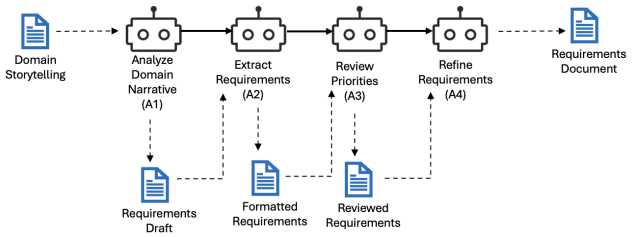

Figure 2: Agents as a workflow in the Requirements graph.

4 RAI's Features

This section provides an overview of RAI's main features by describing how the requirements engineer can navigate the tool and presenting some screenshots. The tool is available at https://rai.leds.dev.br/.

To generate RE artifacts, the requirements engineer must first log into RAI and record the system to each the artifacts will be generated. For that, the user must inform the system's name, description, and modules, select the module to be addressed in the artifacts, and then click the "Start Requirements Specification and Analysis" button that appears in the system/module screen. Then, a pop-up is shown (see Figure 3) and the user must choose "Domain Storytelling" as the first artifact to be generated, upload the video or audio file that will be used as the source, and click the "Create" button. Based on information contained in the file, RAI will generate the domain storytelling. Figure 4 depicts a fragment of the domain storytelling generated for a module of the Quick Library system, which aims to support a library operation and management.

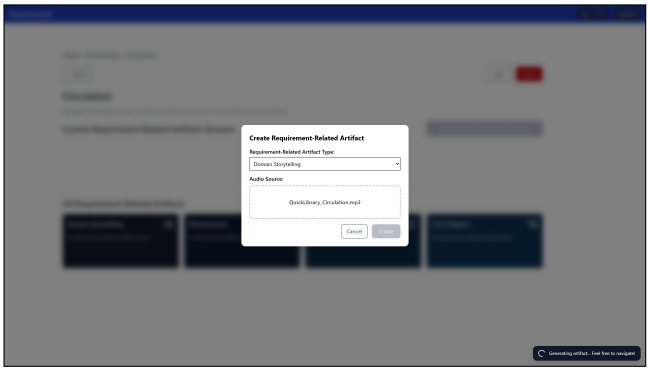

Figure 3: Creating a Domain Storytelling

After generating the domain storytelling, the user can ask the tool to generate the next artifact (Requirements Document) by clicking the light gray button ("Generate Requirements with AI"). This option appears on the screen of every artifact (the button label changes according to the next artifact to be generated) and guides

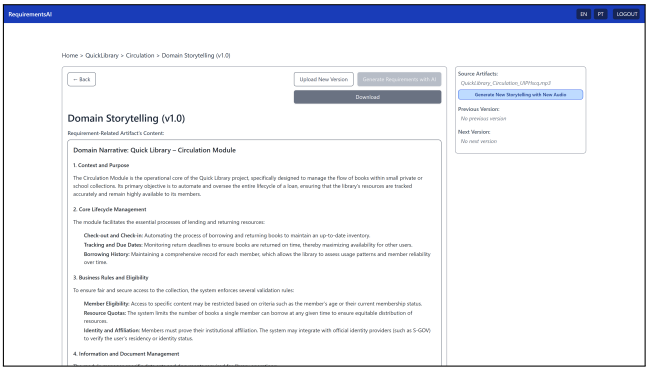

Figure 4: Fragment of a generated Domain Storytelling.

the requirements engineer to the next step. After generating the requirements document, the user can generate the use cases and the class diagram. Figures 5, 6, and 7 depict, respectively, fragments of requirements document, use case, and class diagram artifacts generated automatically by RAI.

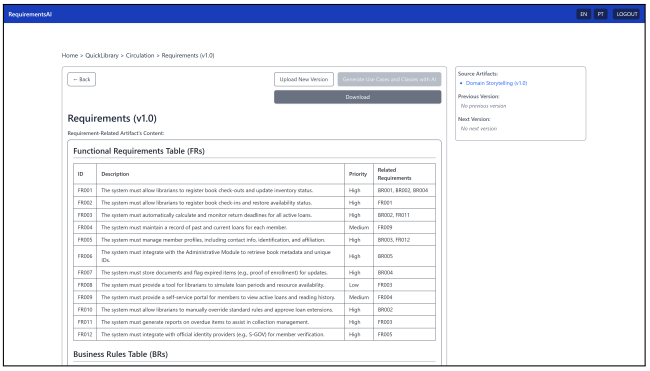

Figure 5: Fragment of a generated Requirements Document

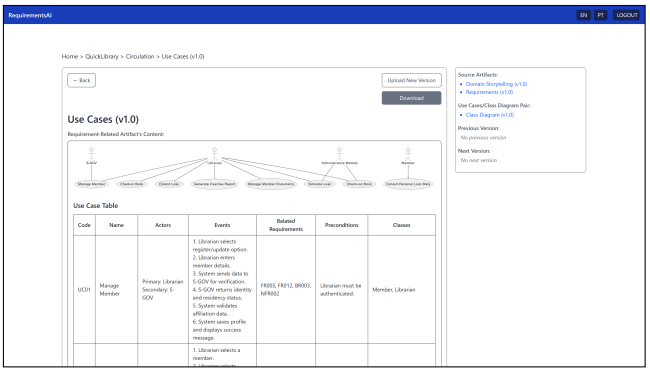

Figure 6: Fragment of generated Use Cases

During the artifacts generation process, the requirements engineer can evaluate and revise the produced artifacts at any time. By using the options provided in the upper right corner of the artifact

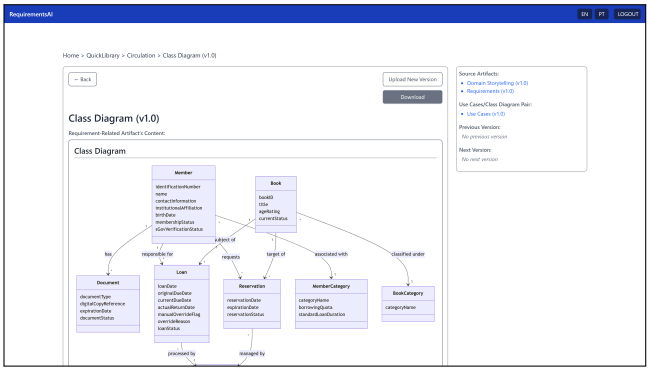

Figure 7: Fragment of generated Class Diagram

screen, the user can download the artifact as a Markdown file, edit it, upload the revised version, and ask the tool to generate the subsequent artifact taking the uploaded revised artifact as the source. Moreover, the requirements engineer can get information about the artifact and the ones related to it. For example, they can see the artifact version, access its source artifact or the artifacts generated from it, as well as previous or later versions of the artifact (when they exist).

The artifacts generated for a system/module are available in the system/module page, as illustrated in Figure 8. In that screen, the produced artifacts are organized in columns according to their type. Different icons and shades of blue are used to differentiate the artifact types. The artifacts are also organized according to their version. The current versions are shown in the first section ("Current Requirement-related Artifacts"). In the second section ("All Requirement-related Artifacts"), all artifacts are available. The user can access an artifact by clicking it in the artifact grid. In Figure 8, there is a single artifact of each type generated for the Circulation module (all the artifacts are in their first version - v1.0).

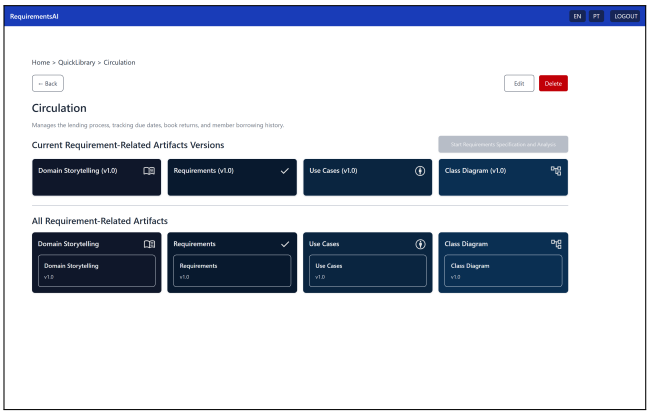

Figure 8: Artifacts related to a module

When a new version of an artifact is produced, either automatically or by the requirements engineer revising/editing an existing artifact and uploading a new version of it, RAI verifies whether there are subsequent artifacts not generated from it and, if so, RAI labels them as "out of sync", indicating that they may be inconsistent. That means that a new version of these artifacts should be generated based on the new version of the source artifact. This mechanism enables the requirements engineer to maintain the history of generated artifacts as well as keep consistency among them. Figure 9 illustrates a situation in which a new version of the domain storytelling artifact (v3.0) was generated from the upload of a new audio file, causing all subsequent artifacts to be "out of sync". The requirements engineer should, thus, generate new versions of the artifacts taking the new domain storytelling as the source. During this synchronizing process, the requirements engineer can indicate (through an option in a pop-up screen presented to them) that new versions are not necessary. For example, if a new version of a requirements document is uploaded by the requirements engineer but the changes made do not impact the use case and class diagram previously generated, the user informs RAI that a new version of use cases and class diagram is not necessary and, thus, RAI links the new version of the requirements document with the use case and class diagram previously generated, considering them synchronized and keeping traceability among related artifacts.

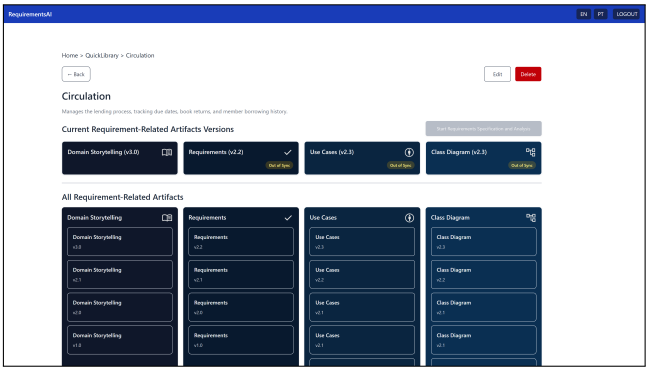

Figure 9: Module with "out of sync" artifacts

5 Implications

Recent advances in AI have changed software development. Software engineers have been required to be more productive and stimulated to use language models to accelerate the development process [12]. Although prompt-based approaches can help, they are reactive, depending on user instructions and providing single-step responses. In the RE context, RAI addresses this limitation by offering a solution that produces RE artifacts automatically. Instead of giving instructions to the LLM, the requirements engineer's role relies mainly on revising and refining the artifacts.

The first version of RAI, which produced two artifacts (domain storytelling and requirements document), was used in a multiple case study in four projects at four organizations. The results indicated that RAI reduced the time and effort required for requirements elicitation and documentation. The participants pointed out that the produced artifacts had good coverage, accuracy, and relevance. They also emphasized the tool's ability to capture implicit information from interviews, organize it into well-structured documents, and generate questions that support domain understanding and

further clarification with stakeholders. These preliminary results indicated that RAI is a promising tool to aid requirements engineers in producing RE artifacts and deepening application domain understanding. It helps accelerate requirements elicitation and documentation, and contributes to producing quality artifacts by identifying open issues and stimulating domain discussions. Information about the case study can be found in the package available in [22].

Given the promising results obtained from using the first version of RAI, we evolved it to its current version. By generating use case and class diagrams, in addition to domain storytelling and requirements document, RAI helps requirements engineers produce several RE artifacts, addressing different system perspectives and, thus, providing a more comprehensive support. By following a human-in-the-loop approach, RAI reinforces the paramount role of the requirements engineer. Thus, requirements engineers can use the tool as a *partner* that helps produce RE artifacts, but requires the requirements engineer to evaluate and refine them by incorporating their expertise into the generation workflow. RAI also contributes to the artifacts' quality by providing information that can help the requirements engineer refine and clarify requirements.

Artifact version management, traceability, and consistency checking are also distinct capabilities of RAI. They enable requirements engineers to evolve the artifacts over the development cycle (e.g., to address stakeholders' needs or changes that occurred during the system development), keeping consistency among related artifacts.

Besides supporting requirements engineers in practice, RAI can also be useful in RE education. It can help students understand how elicitation data can be progressively transformed into structured artifacts. Moreover, it can help them develop critical reflection about requirements. For example, students can produce artifacts manually and then compare to the ones produced by RAI, discussing aspects such as completeness, consistency, ambiguity resolution, and modeling decisions. Furthermore, the tool can help students understand how important is to manage artifact versions and keep traceability among them. Teachers could also use RAI to help prepare teaching materials and practical exercises.

6 Final Considerations

This paper introduced RAI (*Requirements AssIstant*), a LLM-based multi-agent tool that supports requirements engineers in requirements elicitation and documentation. From audio or video inputs, RAI generates four artifacts providing different and complementary perspectives of the system, namely: domain storytelling, requirements document, use cases diagram plus descriptions, and class diagram plus data dictionary. By using the tool, the requirements engineer remains responsible for the artifacts, being able to revise and refine them. RAI also points out open issues, aspects that need clarification, and questions that can help refine the requirements specification. Based on this information, the requirements engineer can investigate the domain further, improve the artifacts, and their understanding of the domain, which is crucial for the next development phases.

Different from prompt-based solutions, RAI provides a pipeline of LLM-based agents that cooperate in a workflow to produce RE artifacts and keeps the requirements engineering as the main role in the process. Preliminary results suggest that RAI has the potential to reduce manual effort, accelerate artifact generation, and contribute to quality by helping identify ambiguities, inconsistencies, and information gaps. The results are promising, but it is important to notice that the quality of the produced artifacts depends directly on the quality of the elicitation recordings. If data is ambiguous and incomplete, the produced RE artifacts will reflect it.

As future work, we intend to evolve RAI features and add new ones. A necessary improvement regards the artifacts' format. Currently, they are generated in Markdown and textual modeling formats, which may limit users not familiar with these technologies from editing the artifacts. We also intend to extend RAI to better support the integration of artifacts produced to different modules of a system. In its current version, RAI addresses each module in isolation, without explicitly addressing dependencies, overlaps, or conflicts among different modules. To address this limitation, we intend to incorporate specialized AI agents capable of verifying connections among modules and identifying intersections, dependencies, and potential conflicts. In addition, we plan to evolve RAI to support an iterative process where multiple information sources (including subsequent meeting recordings) can be used to incrementally update and evolve existing requirements over time. Another direction is to improve the interaction between RAI and the requirements engineer during the automated generation of artifacts, so that inconsistencies, ambiguities, and open questions can be addressed within the generation process itself, rather than only after an artifact has been produced. Additionally, we intend to explore bidirectional synchronization to propagate manual refinements made to downstream artifacts back to the source artifacts.

Given that elicitation data quality affects RE artifacts quality, some advances can be considered to help the requirements engineer gather good data in the elicitation meetings. For example, the development of new agents capable of supporting the requirement engineers in real time during requirements elicitation meetings by suggesting follow-up questions and alerting them to information gaps as the discussion unfolds.

Finally, new studies are necessary to evaluate RAI in practical settings, considering different organizational contexts, project domains, and levels of requirements complexity. These studies will allow us to assess the quality of the generated artifacts, the impact of the tool on requirements engineers' productivity, and its contribution to stakeholder communication and requirements refinement. In addition, studies investigating RAI's applicability and effectiveness in RE education are also needed to evaluate the tool in the educational context.

Artifact Availability

RAI's documentation and source code are available at https://github.com/leds-conectafapes/leds-conectafapes-RequirementsAI.

Acknowledgements

We thank Matheus Lenke Coutinho and Gabriel Zozatelle Borges, who helped in the multiple case study performed to evaluate RAI first version. This research is supported by CAPES (Finance Code 001), FAPES (Processes 370/2026, 2023-N3HJD), and CNPq (Process 304787/2025-6).

References

[1] Sallam Abualhaija, Marcello Ceci, Nicolas Sannier, Domenico Bianculli, Lionel C Briand, Dirk Zetzsche, and Marco Bodellini. 2024. AI-Enabled Regulatory Change Analysis of Legal Requirements. In *2024 IEEE 32nd International Requirements Engineering Conference (RE)*. IEEE, 5–17. doi:10.1109/RE59067.2024.00012

[2] Sathurshan Arulmohan, Marie-Jean Meurs, and Sébastien Mosser. 2023. Extracting Domain Models from Textual Requirements in the Era of Large Language Models. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 580–587. doi:10.1109/MODELS-C59198.2023.00096

[3] Leonardo Banh, Florian Holldack, and Gero Strobel. 2025. Copiloting the future: How generative AI transforms Software Engineering. *Information and Software Technology* 183 (2025), 107751. doi:10.1016/j.infsof.2025.107751

[4] Monalessa Perini Barcellos. 2020. Towards a Framework for Continuous Software Engineering. In *Proceedings of the 34th Brazilian Symposium on Software Engineering* (Natal, Brazil) *(SBES '20)*. 626–631. doi:10.1145/3422392.3422469

[5] Chitrak Biswas and Souvick Das. 2024. ARIA-QA: AI-agent Based Requirements Inspection and Analysis Through Question Answering. *Innovations in Systems and Software Engineering* (10 2024), 1–16. doi:10.1007/s11334-024-00589-8

[6] Maxim Bragilovski, Ashley T Van Can, Fabiano Dalpiaz, and Arnon Sturm. 2024. Deriving Domain Models from User Stories: Human vs. Machines. In *2024 IEEE 32nd International Requirements Engineering Conference (RE)*. IEEE, 31–42. doi:10.1109/RE59067.2024.00014

[7] Lan Cao and Balasubramaniam Ramesh. 2008. Agile Requirements Engineering Practices: An Empirical Study. *IEEE Software* 25, 1 (2008), 60–67. doi:10.1109/MS.2008.1

[8] Victorio Albani Carvalho, Monalessa Perini Barcellos, Rodrigo Calhau, Guilherme Borges Cabral, Nicolas Duarte Botelho, Moisés Savedra Omena, and Paulo Sérgio Santos Jr. 2026. Functional Requirements of RAI: Suplementary material of the paper "RAI: An LLM-Based Multi-Agent Tool for Requirements Engineering Artifacts Generation". https://figshare.com/s/9ba61229061efcb50264

[9] Arghavan Moradi Dakhel, Amin Nikanjam, Vahid Majdinasab, Foutse Khomh, and Michel C. Desmarais. 2024. Effective Test Generation Using Pre-trained Large Language Models and Mutation Testing. *Information and Software Technology* 171 (2024), 107468. doi:10.1016/j.infsof.2024.107468

[10] Marian Daun, Alicia M Grubb, Viktoria Stenkova, and Bastian Tenbergen. 2023. A Systematic Literature Review of Requirements Engineering Education. *Requirements Engineering* 28, 2 (2023), 145–175. doi:10.1007/s00766-022-00381-9

[11] Katia Romero Felizardo, Igor Steinmacher, Márcia Sampaio Lima, Anderson Deizepe, Tayana Uchôa Conte, and Monalessa Perini Barcellos. 2024. Data Extraction for Systematic Mapping Study Using a Large Language Model—a proof-of-concept study in software engineering. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 407–413. doi:10.1145/3674805.3690743

[12] Farnaz Fotrousi and Theocharis Tavantzis. 2024. ReqGenie: GPT-Powered Conversational - AI for Requirements Elicitation. In *International Conference on Product-Focused Software Process Improvement*. Springer, 352–359. doi:10.1007/978-3-031-78386-9_25

[13] Bahar Gezici and Ayça Tarhan. 2025. Explainable AI Framework for Software Defect Prediction. *Journal of Software: Evolution and Process* 37, 4 (2025). doi:10.1002/smr.70018

[14] Reginald Putra Ghozali, Herry Saputra, M Apriadin Nuriawan, Ditdit Nugeraha Utama, Ariadi Nugroho, et al. 2019. Systematic Literature Review on Decision-Making of Requirement Engineering from Agile Software Development. *Procedia Computer Science* 157 (2019), 274–281. doi:10.1016/j.procs.2019.08.167

[15] Binnur Görer and Fatma Başak Aydemir. 2024. GPT-Powered Elicitation Interview Script Generator for Requirements Engineering Training. In *2024 IEEE 32nd International Requirements Engineering Conference (RE)*. IEEE, 372–379. doi:10.1109/RE59067.2024.00044

[16] Haolin Jin, Zechao Sun, and Huaming Chen. 2024. RGD: Multi-LLM Based Agent Debugger via Refinement and Generation Guidance. In *Proceedings of the 2024 IEEE International Conference on Agents (ICA)*. IEEE, 136–141. doi:10.1109/ICA63002.2024.00037

[17] Nataliia Klievtsova, Juergen Mangler, Timotheus Kampik, and Stefanie Rinderle-Ma. 2024. Utilizing Process Models in the Requirements Engineering Process Through Model2Text Transformation. In *2024 IEEE 32nd International Requirements Engineering Conference (RE)*. IEEE, 205–217. doi:10.1109/RE59067.2024.00028

[18] Rainer Lutze and Klemens Waldhör. 2024. Generating Specifications from Requirements Documents for Smart Devices Using Large Language Models (LLMs). In *International Conference on Human-Computer Interaction*. Springer, 94–108. doi:10.1007/978-3-031-60405-8_7

[19] Nuno Marques, Rodrigo Rocha Silva, and Jorge Bernardino. 2024. Using ChatGPT in Software Requirements Engineering: A Comprehensive Review. *Future Internet* 16, 6 (May 2024), 21. doi:10.3390/fi16060180

[20] Ruaridh Mon-Williams, Gen Li, Ran Long, Wenqian Du, and Christopher G Lucas. 2025. Embodied Large Language Models Enable Robots to Complete Complex Tasks in Unpredictable Environments. *Nature Machine Intelligence* (2025), 1–10. doi:10.1038/s42256-025-01005-x

[21] Nathalia Nascimento, Paulo Alencar, and Donald Cowan. 2023. Self-Adaptive Large Language Model (LLM)-Based Multiagent Systems. In *2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. IEEE, 104–109. doi:10.1109/ACSOS-C58168.2023.00048

[22] Moisés Savedra Omena, Paulo Sérgio Santos Jr, Matheus Lenke Coutinho, Gabriel Zozatelle Borges, and Monalessa Perini Barcellos. 2026. Suplementary material of the paper "RAI: An LLM-Based Multi-Agent Tool for Requirements Engineering Artifacts Generation" - Study Package. https://figshare.com/s/dee345e51ab71a69c2a3

[23] Krishna Ronanki, Christian Berger, and Jennifer Horkoff. 2023. Investigating ChatGPT's Potential to Assist in Requirements Elicitation Processes. In *2023 49th Euromicro conference on software engineering and advanced applications (SEAA)*. IEEE, 354–361. doi:10.1109/SEAA60479.2023.00061

[24] Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip. 2023. An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation. *IEEE Transactions on Software Engineering* 50, 1 (2023), 85–105. doi:10.1109/TSE.2023.3334955

[25] Jonathan Silva, Qin Ma, Jordi Cabot, Pierre Kelsen, and Henderik Alex Proper. 2026. Towards Human-in-the-Loop LLM-Enabled Domain Modeling. In *Conceptual Modeling. ER 2025 (Lecture Notes in Computer Science, Vol. 16189)*. Springer. doi:10.1007/978-3-032-08623-5_7

[26] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A Survey on Large Language Model Based Autonomous Agents. *Frontiers of Computer Science* 18, 6 (2024), 186345. doi:10.1007/s11704-024-40231-1

[27] Zheying Zhang, Maruf Rayhan, Tomas Herda, Manuel Goisauf, and Pekka Abrahamsson. 2024. LLM-based Agents for Automating the Enhancement of User Story Quality: An Early Report. In *International Conference on Agile Software Development*. Springer Nature Switzerland, Cham, 117–126. doi:10.1007/978-3-031-61154-4_8